1. Suppose the string "yak" is unlucky. Given a string, return a version where all the "yak" are removed, but the "a" can be any char. The "yak" strings will not overlap.   
     
   stringYak("yakpak") → "pak"  
   stringYak("pakyak") → "pak"  
   stringYak("yak123ya") → "123ya"

Solution:

public String stringYak(String str) {

String result = "";

for (int i = 0; i < str.length(); i++){

if (i + 2 < str.length() && str.charAt(i) == 'y' && str.charAt(i+2) == 'k'){

i = i + 2;

} else{

result = result + str.charAt(i);

}

}

return result;

}

1. Given a string, return true if the first instance of "x" in the string is immediately followed by another "x".   
     
   doubleX("axxbb") → true  
   doubleX("axaxax") → false  
   doubleX("xxxxx") → true

Solution:

boolean doubleX(String str) {

int i = str.indexOf("x");

if (i == -1) return false; // no "x" at all

// Is char at i+1 also an "x"?

if (i+1 >= str.length())

return false; // check i+1 in bounds?

return str.substring(i+1, i+2).equals("x");

}

OR

boolean doubleX(String str) {

int i = str.indexOf("x");

if (i == -1) return false; // no "x" at all

// Another approach -- .startsWith() simplifies the logic

String x = str.substring(i);

return x.startsWith("xx");

}

1. Given a string and a non-negative int n, return a larger string that is n copies of the original string.   
     
   stringTimes("Hi", 2) → "HiHi"  
   stringTimes("Hi", 3) → "HiHiHi"  
   stringTimes("Hi", 1) → "Hi"

Solution:

public String stringTimes(String str, int n) {

String result = "";

for (int i =0; i < n; i++ ){

result = result + str;

}

return result;

}

1. Given a string and a non-negative int n, we'll say that the front of the string is the first 3 chars, or whatever is there if the string is less than length 3. Return n copies of the front;   
     
   frontTimes("Chocolate", 2) → "ChoCho"  
   frontTimes("Chocolate", 3) → "ChoChoCho"  
   frontTimes("Abc", 3) → "AbcAbcAbc"

Solution:

public String frontTimes(String str, int n) {

int frontLen = 3;

if (frontLen > str.length()){

frontLen = str.length();

}

String front = str.substring(0,frontLen);

String result = "";

for (int i = 0; i < n; i++){

result = result + front;

}

return result;

}

OR

public String frontTimes(String str, int n) {

int len = str.length();

String temp = "";

if (len < 4) {

for (int i = 0; i < n; i++) {

temp += str;

}

} else {

for (int j = 0; j < n; j++) {

temp += str.substring(0,3);

}

}

return temp;

}

1. Given an array of ints, return the number of 9's in the array.   
     
   arrayCount9({1, 2, 9}) → 1  
   arrayCount9({1, 9, 9}) → 2  
   arrayCount9({1, 9, 9, 3, 9}) → 3

Solution:

public int arrayCount9(int[] nums) {

int count = 0;

for (int i = 0; i < nums.length; i++)

if ( nums[i] == 9)

count++;

return count;

}

1. Given 2 strings, a and b, return the number of the positions where they contain the same length 2 substring. So "xxcaazz" and "xxbaaz" yields 3, since the "xx", "aa", and "az" substrings appear in the same place in both strings.   
     
   stringMatch("xxcaazz", "xxbaaz") → 3  
   stringMatch("abc", "abc") → 2  
   stringMatch("abc", "axc") → 0

Solution:

public int stringMatch(String a, String b) {

// Figure which string is shorter.

int len = Math.min(a.length(), b.length());

int count = 0;

// Look at both substrings starting at i

for (int i=0; i<len-1; i++) {

String aSub = a.substring(i, i+2);

String bSub = b.substring(i, i+2);

if (aSub.equals(bSub)) { // Use .equals() with strings

count++;

}

}

return count;

}

1. Given an array of ints, we'll say that a triple is a value appearing 3 times in a row in the array. Return true if the array does not contain any triples.   
     
   noTriples({1, 1, 2, 2, 1}) → true  
   noTriples({1, 1, 2, 2, 2, 1}) → false  
   noTriples({1, 1, 1, 2, 2, 2, 1}) → false

Solution:

public boolean noTriples(int[] nums) {

for (int i = 0; i < nums.length; i++){

if(i+2 <= nums.length - 1){

if ((nums[i] == nums[i+1]) &&( nums[i] == nums[i+2]))

return false;

}

}

return true;

}

OR

public boolean noTriples(int[] nums) {

// Iterate < length-2, so can use i+1 and i+2 in the loop.

// Return false immediately if every seeing a triple.

for (int i=0; i < (nums.length-2); i++) {

int first = nums[i];

if (nums[i+1]==first && nums[i+2]==first) return false;

}

// If we get here ... no triples.

return true;

}

1. Given a string, return a version where all the "x" have been removed. Except an "x" at the very start or end should not be removed.   
     
   stringX("xxHxix") → "xHix"  
   stringX("abxxxcd") → "abcd"  
   stringX("xabxxxcdx") → "xabcdx"

Solution:

public String stringX(String str) {

String result = "";

for (int i=0; i<str.length(); i++) {

// Only append the char if it is not the "x" case

if (!(i > 0 && i < (str.length()-1) && str.substring(i, i+1).equals("x"))) {

result = result + str.substring(i, i+1); // Could use str.charAt(i) here

}

}

return result;

}

1. Given an array of ints, return the number of times that two 6's are next to each other in the array. Also count instances where the second "6" is actually a 7.   
     
   array667({6, 6, 2}) → 1  
   array667({6, 6, 2, 6}) → 1  
   array667({6, 7, 2, 6}) → 1

Solution:

public int array667(int[] nums) {

int count = 0;

int i;

for (i = 0; i < nums.length -1 ; i++){

if ((nums[i] == 6 && nums[i+1]== 6)||(nums[i] == 6 && nums[i+1]== 7))

count = count + 1;

}

return count;

}

OR

public int array667(int[] nums) {

int count = 0;

// Note: iterate to length-1, so can use i+1 in the loop

for (int i=0; i < (nums.length-1); i++) {

if (nums[i] == 6) {

if (nums[i+1] == 6 || nums[i+1] == 7) {

count++;

}

}

}

return count;

}

1. Given a non-empty string like "Code" return a string like "CCoCodCode".   
     
   stringSplosion("Code") → "CCoCodCode"  
   stringSplosion("abc") → "aababc"  
   stringSplosion("ab") → "aab"

Solution:

public String stringSplosion(String str) {

String result = "";

for (int i = 0; i < str.length(); i++){

result = result + str.substring(0,i+1);

}

return result;

}

1. Count the number of "xx" in the given string. We'll say that overlapping is allowed, so "xxx" contains 2 "xx".   
     
   countXX("abcxx") → 1  
   countXX("xxx") → 2  
   countXX("xxxx") → 3

Solution:

int countXX(String str) {

int count = 0;

for(int i = 0;i < str.length()- 1; i++){

if (str.substring(i,i+2).equals("xx"))

count++;

}

return count;

}

1. Given an array of ints, return true if .. 1, 2, 3, .. appears in the array somewhere.   
     
   array123({1, 1, 2, 3, 1}) → true  
   array123({1, 1, 2, 4, 1}) → false  
   array123({1, 1, 2, 1, 2, 3}) → true

Solution:

public boolean array123(int[] nums) {

for (int i = 0; i < nums.length - 2; i++){

if (nums[i]== 1 && nums[i+1] == 2 && nums[i+2] == 3)

return true;

}

return false;

}

1. Given an array of ints, return true if one of the first 4 elements in the array is a 9. The array length may be less than 4.   
     
   arrayFront9({1, 2, 9, 3, 4}) → true  
   arrayFront9({1, 2, 3, 4, 9}) → false  
   arrayFront9({1, 2, 3, 4, 5}) → false

Solution:

public boolean arrayFront9(int[] nums) {

int end = nums.length;

if (end > 4)

end = 4;

for (int i = 0; i < end; i++){

if (nums[i] == 9)

return true;

}

return false;

}

1. Given a string, return a new string made of every other char starting with the first, so "Hello" yields "Hlo".   
     
   stringBits("Hello") → "Hlo"  
   stringBits("Hi") → "H"  
   stringBits("Heeololeo") → "Hello"

Solution:

public String stringBits(String str) {

String result = "";

for(int i = 0; i < str.length(); i+=2){

result = result + str.substring(i,i+1);

}

return result;

}

1. Given a string, return the count of the number of times that a substring length 2 appears in the string and also as the last 2 chars of the string, so "hixxxhi" yields 1 (we won't count the end substring).   
     
   last2("hixxhi") → 1  
   last2("xaxxaxaxx") → 1  
   last2("axxxaaxx") → 2

Solution:

public int last2(String str) {

// Screen out too-short string case.

if (str.length() < 2) return 0;

String end = str.substring(str.length()-2);

// Note: substring() with 1 value goes through the end of the string

int count = 0;

// Check each substring length 2 starting at i

for (int i=0; i<str.length()-2; i++) {

String sub = str.substring(i, i+2);

if (sub.equals(end)) { // Use .equals() with strings

count++;

}

}

return count;

}

1. Given a string, return a string made of the chars at indexes 0,1, 4,5, 8,9 ... so "kittens" yields "kien".   
     
   altPairs("kitten") → "kien"  
   altPairs("Chocolate") → "Chole"  
   altPairs("CodingHorror") → "Congrr"

Solution:

public String altPairs(String str) {

String result = "";

//Run i by 4 to hit 0, 4, 8,...

for (int i = 0; i < str.length(); i = i + 4){

//Append the chars between 1 and i + 2

int end = i + 2;

if (end > str.length()){

end = str.length();

}

result = result + str.substring(i, end);

}

return result;

}